IN THE UNITED STATES PATENT AND TRADEMARK OFFICE
BEFORE THE BOARD OF PATENT APPEALS AND INTERFERENCES

In re application of: Jeffrey G. Cheng et al.	Examiner: Philip A. Guyton
Application No.: 10/672,180	Group Art Unit: 2113
Filed: September 26, 2003	Docket No.: 00100.03.0032
For: **METHOD AND APPARATUS FOR MONITORING AND RESETTING A CO-PROCESSOR**	

APPELLANT'S REVISED SUMMARY OF CLAIMED SUBJECT MATTER PURSUANT TO MPEP § 1205.03

Dear Sir:

Appellant received a Notification of Non-Compliant Appeal Brief dated September 15, 2008. Although Appellant respectfully submits that its originally-filed Summary of Claimed Subject Matter complied with MPEP § 1205.02(v), Appellant submits this Revised Summary of Claimed Subject Matter pursuant to MPEP § 1205.03. This Revised Summary wholly replaces Section V of Appellant's originally-filed Appeal Brief, filed September 2, 2008 in the above-identified application.

Appellant further submits that the citations to its Specification are exemplary in nature. Support for Appellant's claim limitations are not limited to the specific citations set forth in this summary.

Respectfully submitted,

Date: October 14, 2008

By: [illegible]
Christopher J. Reckamp
Registration No. 34,414

Vedder Price P.C.
222 N. LaSalle Street
Chicago, Illinois 60601
PHONE: (312) 609-7599
FAX: (312) 609-5005

CHICAGO/#1857601.1

V. REVISED SUMMARY OF CLAIMED SUBJECT MATTER

A host processor is capable of optimizing the processing of graphics data and video information by using a graphics processor and related software. (Specification at ¶ 2; AKA p. 2, ll. 7-17.) Such graphics processing is commonly referred to as graphics rendering. *Id.* The host processor communicates with the graphics processor and effectuates drawing commands and graphics rendering by executing a piece of software called a graphics driver. *Id.* Because of the complexity of hardware and software used in and to support accelerated graphics rendering, graphics processors tend to be susceptible to various rendering errors such as hangs. (*Id.* at ¶ 3; AKA p. 2, l. 18 – p. 3, l. 6.) A hang is where a system or process becomes non-responsive, such as where a processor hangs up due to the processor stalling, locking up, getting stuck in an infinite loop, etc. *Id.* Such hangs may cause data loss or may force operations to halt on an entire system. *Id.*

Conventional methods have been developed to detect errors on an operating system level (e.g., at the graphics driver level), which may be indicative of a hang in a graphics processor. (*Id.* at ¶¶ 4-6; AKA p. 3, l. 7 – p. 4, l. 18.) One such method is to periodically implement a watchdog timer that (a) measures the time spent in a graphics driver by performing a spin-loop and (b) waits for the graphics processor to respond. (*Id.* at ¶ 4; AKA p. 3, ll. 7-17.) The detection of an infinite loop is indicative that the graphics processor is likely hung. *Id.* Once a timeout expires without a response, the operating system either stops the graphics driver and prompts the user to restart the system or unloads the accelerated graphics driver and loads a standards VGA driver to continue graphics rendering operations in a non-accelerated mode. (*Id.* at ¶¶ 4, 6; AKA p. 3, ll. 7-17, p. 4, ll. 6-18.)

Such conventional methods, however, are unable to differentiate between what appears to be a hang in the graphics driver and an actual hang. (*Id.* at ¶ 5; AKA p. 3, l. 18 – p. 4, l. 5.) For example, if a large amount of rendering commands are being processed by a graphics processor, the graphics processor may provide a relatively slow response to the spin-loop. *Id.* In certain instances, such a slow response can be improperly interpreted as a hang in the graphics processor, thereby requiring a system reboot or a switch to non-accelerated graphics rendering. (*Id.* at ¶¶ 5-6; AKA p. 3, l. 18 – p. 4, l. 18.)

The subject matter set forth in Appellant's claims on appeal addresses, among other things, the shortcoming set forth above. For example, Appellant's claim 1 is directed toward a circuit for monitoring and resetting a co-processor comprising:

> a hang detector module operative to detect a hang in the co-processor by <u>detecting a discrepancy between a current state of the co-processor and a current activity of the co-processor</u>; and
>
> a selective processor reset module operative to selectively reset the co-preprocessor without resetting a processor, in response to detecting a hang in the co-processor. (Emphasis added.)

Figure 1 provides an exemplary figure corresponding to the elements of claim 1.

With reference to Figure 1 of Appellant's application, the Specification discloses processor 100 coupled to co-processor 102. (*Id.* at ¶ 33; AKA p. 10, ll. 12 -15.) In one embodiment, processor 100 is a host processor and co-processor 102 is a graphics processor. (*Id.* at ¶ 41; AKA p. 13, l. 16 – p. 14, l. 7.) Processor 100 includes circuit 104 comprising a hang detector module 106 and a selective processor reset module 108. (*Id.* at ¶ 34; AKA p. 10, l. 16 – p. 11, l. 8.) The hang detector module 106 examines co-processor 102 to determine if it is hung. *Id.* Unlike the conventional methods that use a watchdog timer and that may generate false negatives, Appellant's Specification provides for the detection of a hang by detecting a discrepancy between a current state of the co-processor 102 and a current activity of the co-processor 102. (*Id.* at ¶ 49; AKA p. 17, ll. 11-17.) Following the detection of the hang, the hang detection module 106 returns a hang notification signal 110 to the selective processor reset module 108. (*Id.* at ¶ 36; AKA p. 11, l. 17 – p. 12, l. 7.) When the selective processor reset module 108 receives the hang notification signal 110, indicating that a hang has occurred in co-processor 102, the selective processor reset module 108 selectively resets co-processor 102 without resetting the processor 100. (*Id.* at ¶ 37; AKA p. 12, ll. 8 –18.)

In one embodiment, the hang detector module 106 determines a current state of the co-processor 102 by determining if the co-processor 102 is currently executing instructions or busy. (*Id.* at ¶ 35; AKA p. 11, ll. 9-16.) For example, the hang detector module 106 may first check to see if a busy flag (e.g., data in one or more storage elements) is set. (*Id.* at ¶¶ 36, 41; AKA p. 11, l. 17 – p. 12, l. 7, p. 13, l. 16 – p. 14, l. 7.) If a busy flag is not set, the hang detector module 106 returns a no-hang notification signal to the selective processor reset module 108 because a co-processor that is not executing any instructions is unlikely to be experiencing a hang. (*Id.* at ¶¶ 36, 26; AKA p. 11, l. 17 – p. 12, l. 7, p. 8, ll. 9-19.) However, if a busy flag is determined to be set, the hang detector module 106 then determines if a current activity of the co-processor 102 is consistent with the current state of the co-processor 102 (indicative of a no hang condition) or whether there is a discrepancy between the current state and the current activity of the co-processor 102 (indicative of a hang condition). (*Id.* at ¶ 35-36; AKA p. 11, l. 9 – p. 12, l. 7.)

Claim 6 corresponds to the method claim of circuit claim 1. The limitations of claim 6 are set forth below with citations to the Specification and Figures supporting each limitation. The citations made above with respect to claim 1 further support the limitations of claim 6.

> A method of monitoring and resetting a co-processor comprising:
> (FIG. 2, element 200)

> detecting a hang in the co-processor by detecting a discrepancy between a current state of the co-processor and a current activity of the co-processor; and (*Id.* at ¶ 42; AKA p. 14, ll. 8-21; FIG. 2, element 204)
>
> selectively resetting the co-processor without resetting a processor, in response to detecting a hang in the co-processor. (*Id.* at ¶ 43; AKA p. 15, ll. 1-7; FIG. 2, element 206)

Claim 24 corresponds to the memory claim of circuit claim 1. The limitations of claim 24 are set forth below with citations to the Specification and Figures supporting each limitation. The citations made above with respect to claim 1 further support the limitations of claim 24.

> A memory containing instructions executable on a processor that causes the processor to: (FIG. 5, element 500)
>
> detect a hang in a co-processor by detecting a discrepancy between a current state of the co-processor and a current activity of the co-processor; and (*Id.* at ¶¶ 50-51; AKA p. 17, l. 18 – p. 18, l. 12; FIG. 5, elements 100, 502, 506, 504, 110)
>
> selectively reset the co-processor without resetting the processor, in response to detecting a hang in the co-processor. (*Id.* at ¶¶ 50-51; AKA. p. 17, l. 18 – p. 18, l. 12; FIG. 5, elements 508)

In another embodiment, the hang detector module is operative to detect a hang in the co-processor by detecting a discrepancy between a current state of the co-processor and data in one or more storage elements associated with the co-processor, wherein the data in the one or more storage elements represents a current activity of the co-processor. This embodiment, is featured in claim 26 and is reprinted below.

> A circuit for monitoring and resetting a co-processor comprising:
>
> a hang detector module operative to detect a hang in the co-processor by <u>detecting a discrepancy between a current state of the co-processor and data in one or more storage elements associated with the co-processor, wherein the data in the one or more storage elements represents a current activity of the co-processor;</u>

> a selective processor reset module operative to selectively reset the co-processor without resetting a processor, in response to detecting a hang in the co-processor. (Emphasis added.)

For the purpose of clarity (1) Figure 1 remains illustrative of the features set forth in claim 26 and (2) the claimed selective processor reset module is operative to selectively reset the co-processor as described above with respect to claim 1.

In this embodiment, the hang detector module 106 is capable of detecting the current activity of co-processor 102 by examining corresponding registers or other storage elements associated with the co-processor 102 to determine whether activity in those registers reflects the actual processing of instructions. (*Id.* at ¶¶ 35-36; AKA p. 11, l. 9 – p. 12, l. 7.) In one embodiment, corresponding registers are examined before and after a suitable wait period to ensure the co-processor 102 has an opportunity to process the instructions and alter the data in the registers. (*Id.* at ¶ 36; AKA p. 11, l. 17 – p. 12, l. 7.) In this embodiment, a comparison between the two sets of register contents (i.e., on either side of the wait period) can be performed by the hang detection module 106 to detect a difference in the data. *Id.*

If a difference (i.e., activity) is detected, the co-processor 102 is not hung because the current activity of the co-processor 102 is consistent with the current state of the co-processor, and a no-hang notification signal may be set to the selective processor reset module. *Id.* If no difference or activity is detected, the co-processor 102 is hung <u>because a discrepancy between a current state of the co-processor 102 and a current activity of the co-processor 102 has been detected</u>. (*Id.* at ¶ 49; AKA p. 17, ll. 11-17.) A discrepancy is said to exist because the state of the co-processor is "busy" (i.e., the co-processor is executing instructions), but the co-processor's activity is <u>not consistent with</u> the busy state of the co-processor 102 (i.e., the co-processor's registers do not show a difference in stored data). (*Id.* at ¶¶ 35-36, 41, 48-49; AKA p. 11, l. 9 – p. 12, l. 7, p. 13, l. 16 – p. 14, l. 7, p. 17, ll. 1-17.)

Claim 30 corresponds to the method claim of circuit claim 26. The limitations of claim 30 are set forth below with citations to the Specification and Figures following supporting each limitation. The citations made above with respect to claim 26 further support the limitations of claim 30.

> A method of monitoring and resetting a co-processor comprising: (FIG. 2, element 200)
>
> detecting a hang in the co-processor by detecting a discrepancy between a current state of the co-processor and data in one or more storage elements associated with the co-processor, wherein the data in the one or more storage elements represents a current activity of the co-processor; and (*Id.* at ¶ 42; AKA p. 14, ll. 8-21; FIG. 2, element 204)
>
> selectively resetting the co-processor without resetting a processor, in response to detecting a hang in the co-processor. (*Id.* at ¶ 43; AKA p. 15, ll. 1-7; FIG. 2, element 206)

Claim 34 corresponds to the memory claim of circuit claim 26. The limitations of claim 34 are set forth below with citations to the Specification and Figures following supporting each limitation. The citations made above with respect to claim 26 further support the limitations of claim 34.

> A memory containing instructions executable on a processor that causes the processor to: (FIG. 5, element 500)
>
> detect a hang in a co-processor by detecting a discrepancy between a current state of the co-processor and data in one or more storage elements associated with the co-processor, wherein the data in the one or more storage elements represents a current activity of the co-processor; and (*Id.* at ¶¶ 50-51; AKA p. 17, l. 18 – p. 18, l. 12; FIG. 5, elements 100, 502, 506, 504, 110)

> selectively reset the co-processor without resetting the processor, in response to detecting a hang in the co-processor. (*Id.* at ¶¶ 50-51; AKA. p. 17, l. 18 – p. 18, l. 12; FIG. 5, elements 508)

Claim 35 sets forth an embodiment, wherein "the discrepancy is detected by comparing data representing a current state of the co-processor with data representing a current activity of the co-processor." As mentioned above with respect to claim 1, the data representing a current state of the co-processor may, in one embodiment, be termed a busy flag. (*Id.* at ¶¶ 36, 41; AKA p. 11, l. 17 – p. 12, l. 7, p. 13, l. 16 – p. 14, l. 7.) The data representing a current state of the co-processor is compared to data represent activity of the co-processor. The current activity of the co-processor may be determined as set forth above with respect to claim 26. If a current activity of the co-processor 102 is not consistent with the current state of the co-processor 102, a hang condition is determined. (*Id.* at ¶ 35-36; AKA p. 11, l. 9 – p. 12, l. 7.)

Claim 11 requires a circuit for monitoring and resetting a co-processor comprising:

> a hang detector module operative to detect a hang in a co-processor;
>
> a <u>halt communications module operative to halt executable instruction communications with the co-processor</u>, in response to detecting a hang in the co-processor;
>
> a selective processor reset module operative to selectively reset the co-processor without resetting a processor, in response to detecting a hang in the co-processor;
>
> a reset check module operative to detect if the co-processor has been successfully reset, in response to the resetting of the co-processor; and
>
> a restart communications module operative to restart executable instruction communications with the co-processor, in response to detecting that the co-processor has been successfully reset. (Emphasis added.)

Figure 3 is illustrative of this embodiment. It is appreciated that the hang detector module and selective processor reset module of claim 11 and of Figure 3 are as generally described above with respect to claim 1. Additionally, circuit 104 may include, among other things, a halt communications module 300. (*Id.* at ¶ 44; AKA p. 15, ll. 8-14.) Halt communications module 300 is used to halt command communications with co-processor 102 in response to receiving hang notification signal 110 from hang detector module 106 indicating a hang in the co-processor 102. The halt communications module 300 may be used to stop the sending of instructions to the co-processor 102. (*Id.* at ¶ 45; AKA p. 15, l. 15 – p. 16, l. 9.) In one embodiment, halt communications module 300 may implement this task by setting a send flag, a receive flag and/or a busy flag, each associated with communication between the processor 100 and the co-processor 102, to "off". *Id.* By setting a send flag to "off", processor 100 is prevented from sending instructions to the co-processor 102. *Id.* By setting a receive flag or busy flag to "off", processor 100 does not expect to receive any return signals, data or the like from co-processor 102. *Id.*

In response to receiving a reset notification signal 306 from the selective processor reset module 108, the reset check module 302 determines if the reset was successful. If successful, the reset check module 302 generates a reset success notification signal 308 and sends it to a restart communications module 304. (*Id.* at ¶ 46; AKA p. 16, ll. 10-14.) The restart communications module 304 is responsive to the reset success notification signal 308 to restart executable instruction communications with the co-processor. (*Id.* at ¶ 47; AKA p. 16, ll. 15-21.)